.NET Technologies 2006

University of West Bohemia
Campus Bory

May 29 – June 1, 2006

Posters papers proceedings

Edited by

Jens Knoop, Vienna University of Technology, Austria
Vaclav Skala, University of West Bohemia, Czech Republic

.NET Technologies – Poster papers conference proceedings

Editor-in-Chief:	Vaclav Skala University of West Bohemia, Univerzitni 8, Box 314 306 14 Plzen Czech Republic *skala@kiv.zcu.cz*
Managing Editor:	Vaclav Skala

Author Service Department & Distribution:

Vaclav Skala - UNION Agency
Na Mazinach 9
322 00 Plzen
Czech Republic
Reg.No. (ICO) 416 82 459

Hardcopy: ***ISBN 80-86943-12-7***

Contents

Knowledge.NET ontology-based knowledge management toolkit for Microsoft.NET

Vladimir Safonov
St. Petersburg university
28 Universitetsky prospect
Petrodvorets
St. Petersburg 198504 Russia
v_o_safonov@mail.ru

Anton Novikov
St. Petersburg university
28 Universitetsky prospect
Petrodvorets
St. Petersburg 198504 Russia
antonnovik@gmail.com

Alexey Smolyakov
St. Petersburg university
28 Universitetsky prospect
Petrodvorets
St. Petersburg 198504 Russia
smlkvalex@mail.ru

Dmitry Cherepanov
St. Petersburg university
28 Universitetsky prospect
Petrodvorets
St. Petersburg 198504 Russia
hail@pochtamt.ru

Maxim Sigalin
St. Petersburg university
28 Universitetsky prospect
Petrodvorets
St. Petersburg 198504 Russia
max_ilya@mail.ru

ABSTRACT

The integration of knowledge engineering to software engineering is one of the most promising software development methodologies. It enables the combination of "traditional" programming features with knowledge engineering constructs in one intelligent solution, which is desirable in modern software development.

Most of knowledge engineering tools are isolated, limited and based on very specific languages such as clones of LISP[Car60a] or PROLOG[Der96a]. As such, they are prevented from integrating them with more conventional languages and application packages.

In contrast, our Knowledge.NET knowledge management toolkit for Microsoft.NET is based on a hybrid knowledge representation language, an extension of C# with knowledge engineering features – ontologies and rule sets.

The system can be used to develop knowledge libraries (bases) and intelligent solutions for a variety of problem domains. The knowledge can be developed using the Knowledge.NET system knowledge editor, or can be extracted from the Internet or any text files.

The system also provides a knowledge converter from Knowledge.NET format to more commonly used KIF (Knowledge Interchange Format).

Keywords
Microsoft.NET, knowledge management, ontology, rule set, C#, converter, knowledge extractor, knowledge editor, Knowledge Interchange Format, knowledge converter, Microsoft Visual Studio.NET 2005, add-in

1. INTRODUCTION

Knowledge engineering originated in 1950s as a part of Artificial Intelligence (AI). In general, it studies how to create expert systems that solve "creative" problems in certain problems domain [Saf91a]. In reality, many problems can't be resolved by pure algorithms only. Problem solving can be also related to some hierarchical (conceptual), factual and heuristic knowledge.

Some of the popular methods of knowledge representation listed below:

- Productions – represents knowledge as a set of rules: IF condition THEN action. Rules are convenient for representing heuristic knowledge. Nowadays, PROLOG is the most common language to build rule-based expert systems;
- Frames – hierarchical knowledge structures. A frame has a set of slots and can be inherited from other frames. In turn, each of the slots has a value which

can be a value of simple type as well as a reference to another frame. The concept of frame was introduced by Marvin Minsky in 1970s [Min74a].

- Ontologies. The term *ontology* was borrowed from philosophy in early 1990s. The most known definition of ontology was formulated by Tom Gruber: "An ontology is an explicit specification of a conceptualization". More specifically, we define the ontology as a kind of specification of a problem domain in terms of its concepts and their relationships that allows to describe and to share hierarchical and factual knowledge in very efficient way. Knowledge Systems Laboratory of Stanford University supports visualization tool Protégé [Tut04a], which allows to manage ontology-based and frame-based knowledge.

Usually, knowledge engineering tool supports only one type of knowledge and has a very specific and limited semantic of base language, e.g. LISP, PROLOG. In our opinion, it increases complexity of usage of these tools. In contrast to many other tools, Knowledge.NET supports hybrid knowledge (productions, ontologies). More of that, as a base language, we use C# [Spe05a] extended by additional keywords and constructs for knowledge representation.

In section 2 we briefly describe principles of the Knowledge.NET system. In section 3 we present our approach to convert knowledge into KIF format. Section 4 briefly describes knowledge extractor subsytem. The summary section concludes and outlines the perspectives for the future.

2. PRINCIPLES OF KNOWLEDGE.NET

Seamless Integration to Visual Studio

Like Aspect.NET, the Knowledge.NET system is implemented as an add-in to Visual Studio.NET 2005. It actually means that, on installing the add-in, Knowledge.NET GUI becomes part of Visual Studio GUI. Due to that, it is possible to use all of VS.NET's wide spectrum tools and features for application design, implementation, debugging, profiling, etc., when developing an intelligent solution in Knowledge.NET. The add-in contains the following components:

- Knowledge Editor and Coloring;
- Coloring and IntelliSense for Knowledge.NET language;
- Add new type of projects: Knowledge.NET

Hybrid Knowledge Representation Language

The Knowledge.NET language is an extension of C# by constructs for representing hybrid, ontology-based knowledge. Semantics of ontology in Knowledge.NET is similar to that of the OWL knowledge representation language [Ove04a] developed by the W3C consortium.

A Knowledge.NET source code is at first converted into the ordinary C# code, and then compiled by the .NET C# compiler into an assembly that can be used as any other .NET application.

2.1.1 Program structure

A Knowledge.NET application source code consists of the following parts

- C# source code;
- Knowledge .NET – specific source code (concepts, properties, instances, rule sets)

The C# source code part is separated from Knowledge .NET source code by the "#ontology" keyword.

The user can use from her C# code concepts and their properties using standard way of addressing: [concept_name].[property_name]

Query language

The Knowledge.NET query language is one of the ways of accessing its knowledge base. It allows to select from ontology instances satisfying a given criterion (query).

As an example, consider a query on the ontology "Vehicles": "All the vehicles whose maximal speed is not more than 100 kilometers per hour":

Individuals of concepts Vehicle where HasMaximumSpeed <= 100

The result of the above request will be a set of instances of the Vehicle concept whose HasMaximalSpeed property's value is not greater than 100 km/hour.

The query language is supported by the QueryEngine class defined in the Knowledge.NET class library.

Rule sets

Besides conceptual knowledge representation, Knowledge.NET also allows to define heuristic knowledge using rule sets. Thus, the above mentioned hybrid knowledge framework is supported. The format of rule set is very close to that used in the KEE system. As a context of a rule set, an ontology is used which contains that rule set. To use rules, Knowledge.NET contains a classical style implementation of forward and backward chaining. If necessary, one can use her own inference engine by implementing the IProductionSystem interface.

Knowledge Editor

Besides the Knowledge.NET language, we also developed a Knowledge Editor (see fig. 1). The editor can be used to browse, update and enter knowledge. So, the knowledge engineer can work either in interactive mode or by creating knowledge directly in Knowledge.NET source. The knowledge editor also allows to navigate from graphical knowledge representation to textual knowledge representation and to call the converter from Knowledge.NET into C#.

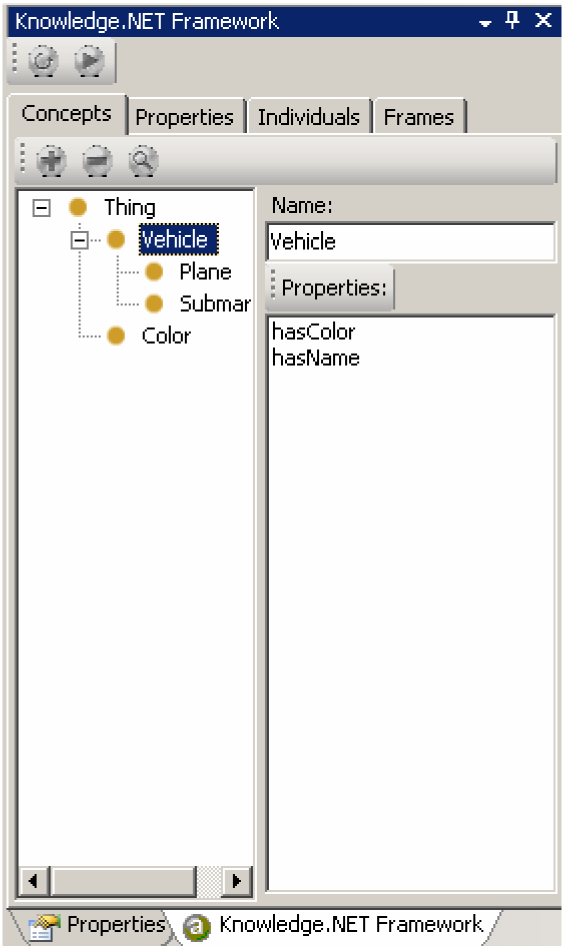

fig. 1

3. SUPPORTING FOR THE KIF FORMAT

Knowledge.NET also contains a converter of knowledge from Knowledge.NET to KIF (Knowledge Interchange Format) [Spe98a], to make the knowledge created in Knowledge.NET available for experts working in more traditional KIF format.

The conversion process consists of two stages. At the first stage the input document in Knowledge.NET is parsed and its internal representation is generated, in the format of the well known Ontolingua [Man97a] knowledge representation language. Actually at this stage the conversion process can be finished: the user can now use the output document in Ontolingua format. This approach is convenient because the user may convert this document into any of the knowledge representation languages supported by Ontolingua (in particular, to use the Ontolingua's KIF compiler).

At the second stage, the knowledge is converted into KIF format.

Overview of Ontolingua.

Ontolingua is a knowledge engineering environment containing a set of functions to work with ontologies (browsing, creating, updating and using). The Ontolingua language is a superset of KIF and contains constructs to represent frames and ontologies. The system also has a number of translators into other knowledge representation languages – Loop, Epikit, Express, Generic-Frame, Algernon, IDL and into KIF [Gru93a].

The Conversion Process

At the first stage, frames and concepts are converted into structures corresponding to Ontolingua's ***define-class*** and ***define-instance*** (for instances), properties – to ***define-relation***.

At the second stage, the following happens:

1. All the used relations are determined. Their representation in terms of KIF is written to the output stream.

2. define-relation and define-class are transformed into the DEFRELATION construct of the KIF language the following way:

:IFF-DEF replaced by ":="

:DEF replaced bt ":=>"

:CONSTRAINTS replaced by "=>"

:EQUIVALENT replaced by "<=>"

:AXIOM-DEF replaced by :AXIOM

3. For all primitive numbers used in the input Knowledge.NET source, the appropriate relations are generated (using the *defrelation* construct). The

following two KIF built-in numeric relations are used:

`integer` - expression`(integer t)` denotes that the *t* object is an integer;

`real` and `(real t)` – the same for real numberss.

For example, the byte type can be converted as follows:

(defrelation byte (?x) :=
  (and (natural ?x) (<= ?x 255)))
(defrelation natural (?x) :=
  (and (integer ?x) (>= ?x 0)))

4. DATA MINING

Knowledge.NET contains a knowledge extractor. This subsystem can be used to generate "raw" ontology from an array of documents or from some repository (e.g., from the Internet).

The scheme of functioning of the knowledge extractor is as follows:

- Morphological analysis of the input text and constructing a set of entities;
- Semantic analysis of the set of entities and constructing the knowledge graph;
- Analysis of the knowledge graph.

Morphological Analysis

At this stage, each word is converted to its *normal form* using MRD dictionaries and XML dictionaries.

Semantic Analysis

At this stage, a knowledge graph is constructed, with the nodes representing entities and edges representing relations. Analysis of entities is performed by customizable templates written in a specialized macro definition language.

Analysis of the Knowledge Graph

At this concluding stage:

- Different properties can be unified into one class. For example, the properties "big" and "giant" can be united into one class
- The graph is cleaned up to delete obsolete relations. For example, if an ascendant class has some property, the same property can be omitted from its descendant classes.

5. SUMMARY

Currently the work on Knowledge.NET is at the initial stage – the first prototype is developed and will be used to solve a variety of practical tasks that require using both "traditional" programming and knowledge engineering.

In future, we plan to integrate Knowledge.NET with Aspect.NET [Saf05a], to enable using Knowledge.NET for aspect-oriented knowledge engineering.

6. REFERENCES

1. [Gru93a] Gruber, T.R. A Translation Approach to Portable Ontology Specifications, 1993
2. [Man97a] Ontolingua Reference Manual at http://www-ksl.stanford.edu/htw/dme/thermal-kb-tour/ontolingua.html
3. [Ove04a] OWL Web Ontology Overview at http://www.w3.org/TR/owl-features/
4. [Saf91a] Safonov, V.O. TIP technology and its application in developing compilers and expert systems for high-performance computing systems. Doctoral dissertation, Leningrad, 1991
5. [Saf05a] Safonov, V.O. Aspect.NET: Aspect-Oriented Programming for Microsoft.NET in Practice. NET Developer's Journal, July 2005
6. [Spe98a] KIF language specification at http://logic.stanford.edu/kif/dpans.html
7. [Spe05a] C# Language Specification at http://msdn.microsoft.com/vcsharp/programming/language/#Language%20Specifications
8. [Tut04a] Protégé OWL Tutorial at http://www.co-ode.org/resources/tutorials/ProtegeOWLTutorial.pdf
9. [Min74a] Minsky, M. A Framework for Representing Knowledge, MIT-AI Laboratory Memo 306, 1974
10. [Car60a]McCarthy, J. Recursive Functions of Symbolic Expressions and Their Computation by Machine, Part I, April 1960
11. [Der96a]Deransart, P., Ed-Dbali, A., and Cervoni, L. Prolog: The Standard, 1996

Implementing a mobile agent infrastructure on the .NET framework

Antonio Boccalatte, Alberto Grosso, Christian Vecchiola
DIST – University of Genoa
Via Opera Pia 13
16142, Genova, Italy
{nino, agrosso, christian}@dist.unige.it

ABSTRACT

This paper presents the solution adopted by the AgentService platform in implementing a software infrastructure for mobile agents. The mobility service takes advantage of the agent model provided by the platform which offers the separation between the state of the agent and its activities. The modular architecture of the platform allows an elegant integration of the mobility service whose implementation resides within an additional platform module. The mobility infrastructure offers its services to the agents that can be stopped, moved, and restarted in a transparent manner. AgentService provides a sort of weak mobility service: during a transfer the state of the agent is maintained while the activities it performs are started from the beginning. The mobility infrastructure is a component that enriches the platform features and allows the implementation of more complex services such as load balancing strategies among different AgentService installations.

Keywords
Agent Mobility, Load Balancing Policy, Agent Framework

1. INTRODUCTION

Software mobility is a software property which can bring robustness, performance, scalability or expressiveness to systems [Kar98a]. Code mobility concerns the ability to migrate a unit of running code from one host to another by preserving partially or totally its execution state [Cab00a]. In particular, systems that completely maintain the execution state are said to support *strong mobility*, while systems that discard the execution state are said to provide *weak mobility*. The possibility of moving running code among computing environments is an interesting opportunity for dynamic intelligent agent systems. Agents are autonomous, pro-active, and socially able: the ability to move and to migrate between different nodes of the community enhances the previously cited features. Hence, as for objects, mobility is an interesting property for agents and mobile agents have emerged as a paradigm for structuring distributed applications.

Journal of .NET Technologies
Copyright UNION Agency – Science Press,
Plzen, Czech Republic

A definition which sufficiently characterizes the essence of a mobile agent system has been proposed by Chen and Nwana [Che95a, Nwa96a]: "..*a mobile agent is a software entity which exists in a software environment. It inherits some of the characteristics of an agent. A mobile agent must contain all of the following models: an agent model, a life-cycle model, a computational model, a security model, a communication model and finally a navigation model.*".

The idea of mobile agent that cannot be implemented without providing multi-agent systems with a software infrastructure that allows the transfer of agents in a transparent manner. This paper describes the solution adopted within the AgentService framework to implement mobility for agents. The proposed solution turns out to be very effective thanks to the agent model adopted by the framework which separates the state and the behavior of a software agent. By using serialization and reflection the state of the agent and some information of the active behavior of the agent are persisted and transferred to the target site. In the following, the authors will illustrate the main features of the AgentService framework and will explain in detail the architecture and the implementation of the mobility infrastructure. A use case will also explain how to use the mobility infrastructure to implement more complex services such as load balancing among different AgentService installations.

2. AGENTSERVICE

AgentService Main Features

The agent oriented paradigm [Wol99a] can be a useful abstraction to model open and dynamic communities. An agent is an autonomous software entity provided with some levels of "intelligence". Moreover, by means social ability, agents can enhance their performances by interoperating in communities called multi-agent systems (MASs) [Wei99a]. A widely accepted architecture specification for multi-agent systems with a reference agent model is the one proposed by the Foundation of Intelligent Physical Agents (FIPA) [Fip01a].

AgentService [Boc04a] is a framework for developing multi-agent systems based on the Common Language Infrastructure, whereof .NET framework is one implementation. AgentService provides a specific agent model and a runtime environment for agent execution compliant with the FIPA specifications. In literature there are many works concerning agent platforms, the most interesting and known are Zeus [Nwa98a], FIPA-OS [Pos00a], and JADE [Bel99a]; AgentService is characterized by an extremely modular architecture and a flexible agent model which allows the implementation of different agent architectures.

Two different kinds of modules have been designed to model all the features of the AgentService platform: core modules and additional modules. Core modules implement all the services required by the platform instance to set up its activity; they involve management of assemblies in which agent templates are defined (Storage Module), the messaging service, the management of the agent's persistence, and a logging service. Through additional modules new features can be added to the platform: they enrich the platform capabilities but they are not essential for the standard activity of agents.

Within AgentService, agents are designed as software entities whose activity is defined by a particular managed set of data (Knowledge objects) and performed by a set of concurrent behaviors (Behavior objects). A Knowledge object represents a set of correlated data modeling a structured concept of the problem domain. A collection of specified Knowledge objects define the state of a software agent: it can be persisted and portions of it can be shared among the different Behavior objects. Behavior objects contain all the agent computational logic and define the agent aggregate behavior. Behavior objects are concurrent and share the information they need by means of the Knowledge objects. Such distinction between activities and data allows a clear decomposition of the agent definition, represents a flexible and generic model. From the implementation point of view every agent instance is deployed in a dedicated Application Domain, which ensures the autonomy and safety of the code executed inside it (agent activities).

3. MOBILITY IN AGENTSERVICE

Introduction

According to the definition given by Chen and Nwana [Che95a, Nwa96a] a mobile agent inherits all the properties of a software agent and, in addition, contains a navigation model which embraces all aspects of agent mobility from the discovery and resolution [Whi95a] of destination hosts to the manner in which a mobile agent is transported. Hence, the introduction of a navigation model implies the extensions of the agent model defined within AgentService with the previously discussed features.

It can be observed that the ability of identifying and discovering destination hosts is already implemented by using the directory services of the platform: AgentService platforms can join together and define a federation. A federation defines the boundaries into which the mobility service takes place. In the following the authors will focus the attention on the second element characterizing the navigation model that is the machinery required to transfer agents. In order to move an agent the model defining its life cycle needs to be extended with an additional state which characterize the agent while is being moved. As suggested by the FIPA specifications the common life cycle of an agent has been extended by adding the *transit* state and two actions to enter and leave that state (*move* and *execute*). The agent itself can require the move action while the platform, through the Agent Management System (AMS), is responsible of completing the migration by performing the execute operation.

In addition, mobile agents require a suitable runtime environment which provides a transfer service allowing them to move from one node to another: this environment is built on top of a host system. Within AgentService, this runtime provides to the agent with a transfer service based on a variation of weak mobility: even if the execution does not continue exactly by executing the next instruction a partial resume of the execution state has been implemented. The main idea is to exploit the adopted agent model and move just the agent state (knowledge objects) among platforms. Within the target platform the agent activities can be restarted taking advantage of the persisted agent state. In addition, the framework provides developers with an entry point, the *Resume* method, for checking the state and the activities of the agent before it continues the execution.

Architecture of the Mobility Service

The architecture of the mobility service takes full advantage from the agent model adopted by the platform: the separation among the agent state from the activities it performs makes the migration process simple. In order to run an agent the runtime environment needs the information defining the agent state and the assemblies containing the agent definition. Hence, moving an agent among AgentService installations requires moving its state and ensuring the presence of that agent type definition on the target platform. Once an agent is moved it is possible to restart its activity by instantiating a new agent of that specific type and restoring its state. State restoration involves loading the transferred knowledge objects and the activation of all the behaviors objects running when the agent was stopped. The information about knowledge objects and the state of each behavior (ready, active, suspended) are all what is really needed to move an agent.

The process which transfers an agent is activated by a request and can be described as follows:

1. *negotiate*: the AMS of the source target contacts the AMS of the target platform and asks if the agent can be moved;
2. *stop and persist*: if the agent can be moved, the AMS stops its activity, persists its state, and puts it into the transit state (*move* action);
3. *transfer*: the AMS instruct the mobility module to move the agent. The state of the agent is transferred to the target platform. This operation may require the transfer of the assemblies describing the types of the agent or used by it;
4. *restore*: the mobility module notifies the AMS that the transfer is completed. The AMS creates an instance of the same type of the agent received sets its state and invokes the *Resume* method allowing the programmer to customize the re-activation of the agent;
5. *execute*: the agent changes its state from transit into its original state, the AMS of the source platform is notified of the successful transfer and the agent is activated (execute action).

This process is implementation independent and AgentService defines an interface (*IMobilityModule*) that every module that wants to offer this service must implement. In this way, developers can implement the service as they prefer: a web service, an ftp service, or a custom channel.

The model adopted by AgentService to define an agent greatly simplifies the work of the mobility module. Thanks to the clear separation among the state and the activities the maintenance of the execution state is obtained by saving all the knowledge objects composing the knowledge base of the agent and the status of the behavior objects. When the agent is restored the information saved are loaded into the new instance and all the activities previously stopped are started. In particular, the mobility infrastructure has to deal with the transfer of assemblies if the repositories of the two platforms are not synchronized.

The entire process that allows mobility of agents takes place if and only if the platforms are allowed to transfer agents; otherwise it stops the *negotiation* phase. The AMS of the source platform will look in the platform configuration to determine if it is allowed exporting agents, while the AMS of the target platform will check if it is allowed to import agents. These information are contained in the platform profile and administrators can customize the platform behavior by modifying the profile in the configuration file.

4. LOAD BALANCING POLICY

Load balancing in AgentService is managed by the Load Balancing Policy (LBP) module. It provides a service that federates platform instances and creates a unique environment in which agents can move. By default, LBP comes with two policies. The first policy balances the number of agent among platforms, while the second is based on the number of exchanged messages moving in the same platform the agents interacting more frequently. The *platform context*, provided by AgentService to each module, gives access to these information. LBP modules, installed on different nodes, cooperate to constitute a federation of platforms defining the border within the balancing policies can be applied. The federation system adopts a client/server model: each node provides its platform profile to the master node which maintains the federation and applies policies. The LBP configuration file defines the structure of the federation indicating which node works as server. At runtime new platforms can dynamically join the federation by registering their profiles to the master node. The master node applies balancing policies every time an interesting event occurs (i.e. the creation of a new agent, the registration of a new platform with the federation). Developers can implement new load balancing policies and dynamically load them in the server LBP module as plug-ins. Hence only the LBP module of the master node handles the balancing policy.

5. CONCLUSIONS

The AgentService modular architecture allows the design and the implementation of additional services which are fully integrated with the standard ones. The mobility module provides a weak mobility service even if it automatically maintains the agent state. Developers are provided with facilities for customizing agent resumption. The presence of a software mobility infrastructure allows the platform to be enriched with more capabilities. The load balancing policy (LBP) module provides an interesting service for resources management exploiting the mobility service. The LBP module allows administrators to apply load balancing algorithms to a federation of AgentService platforms. The agent transfer process was tested applying the default balancing policies. The tests pointed out that the most onerous operation during the mobility process is the transfer of assemblies required for agent activities. This is a minor drawback since in common balancing scenarios all the nodes are likely to run the same agent types; hence there is no need for huge assembly transfers.

The mobility infrastructure suffers from the lack of interoperability with different FIPA compliant platforms, in particular Jade. The development of an interoperable mobility infrastructure is a very challenging task which involves a lot of problems due to the adoption of different technologies, architectures, and agent models.

6. REFERENCES

[Kar98a] Karnik, N. M., and Tripathi, A. R. Design Issues in Mobile-Agent Programming Systems. IEEE Concurrency 6(3), pp. 52-61, July-September 1998.

[Cab00a] Cabri, G., Leopardi, L., and Zambonelli, F. Weak and Strong Mobility in Mobile Agent Applications. Proceedings of the 2nd International Conference and Exhibition on The Practical Application of Java (PA JAVA 2000), Manchester (UK), April 2000.

[Che95a] Chess, D., Harrison, C., and Kershenbaum. A. Mobile Agents: Are they a good idea?. Technical Report, IBM T.J. Watson Research Center, NY, March 1995.

[Nwa96a] Nwana, H. Software agents: An Overview, Knowledge and Engineering Review 11(3), November 1996.

[Wol99a] Wooldridge, M. Intelligent Agents. In Multi-agent Systems – A Modern Approach to Distributed Artificial Intelligence, G. Weiss Ed., Cambridge, MA, pp. 27-78, 1999.

[Wei99a] Weiss, G. Multi-agent Systems – A Modern Approach to Distributed Artificial Intelligence, G. Weiss Ed., Cambridge, MA, 1999.

[Boc04] Boccalatte, A., Gozzi, A., Grosso, A., and Vecchiola, C. AgentService. The Sixteenth International Conference on Software Engineering and Knowledge Engeneering (SEKE'04), Banff Centre, Banff, Alberta, Canada 20-24 June 2004.

[FIP01a] FIPA Abstract Architecture Specification, http://www.fipa.org/specs/fipa00001/

[Nwa98a] Nwana, H.S., Ndumu, D.T., and Lee, L.C. ZEUS: An advanced Tool-Kit for Engineering Distributed Multi-Agent Systems. Proceedings of PAAM98, pp. 377-391, London, U.K., 1998.

[Pos00a] Poslad, S., Buckle, P., and Hadingham, R. The FIPA-OS agent platform: Open Source for Open Standards, PAAM2000, Machestor, UK, April 2000.

[Bel99a] Bellifemine, F., Rimassa, G., Poggi, A. JADE - A FIPA-compliant Agent Framework. Proceedings of the 4th International Conference and Exhibition on The Practical Application of Intelligent Agents and Multi-Agents, London, 1999.

[Whi95a] White J.: The foundation of the electronic market place. General Magic white paper 1995.

RMI Object Consistency Maintenance Techniques at Distributed Computing

Seong Eun Chu
Department of Computer Science
Chonnam National University
300 Yongbong-Dong
Bug-Gu
500-757, Gwangju, Korea
sechu@jnu.ac.kr

Jae Nam Kim
Department of Digital Animation
Kwangju Women's University
165 Sanjeong-Dong
Gwangsan-Gu
506-713, Gwangju, Korea
jnkim@mail.kwu.ac.kr

Dae Wook Kang
Department of Computer Science
Chonnam National University
300 Yongbong-Dong
Bug-Gu
500-757, Gwangju, Korea
dwkang@jnu.ac.kr

ABSTRACT

Various object caching techniques between a client and a server have been proposed at distributed computing. However, these techniques have handled data consistency only. They have not handled object own consistency. In this paper, we proposed two techniques for object consistency maintenance. The first technique makes it possible that the client confirms the object update time in the server based on RMI (Remote Method Invocation). The second is that the server broadcasts invalid message to the clients. Both techniques are evaluated experimentally, and results show that they could be applied selectively at the distributed applications considering object update frequency.

Keywords
RMI (Remote Method Invocation), Consistency Maintenance, Distributed Computing.

1. INTRODUCTION

New objects are created in server and methods of the objects are invoked by clients of the different memory address in distributed computing. There are several techniques such as RMI, CORBA, DCOM, EJB for the development of object oriented distributed applications. RMI does not need to install separately the middleware for supporting information communication between objects or components because it has been implemented already at JVM. Therefore, RMI has been used widely at distributed computing environments as the most simple object communicational model [Dow98]. A data caching at distributed computing environments has been used efficiently when the frequency of data access is high. Generally, caching is simple, but many elements should be considered to maintain consistency of caching.

Existing techniques about consistency maintenance have been classified by detection-based and avoidance-based categories. Detection-based algorithms permit stale data in local, examine the validity of a server when read or write operations are performed. These algorithms maintain the consistency by sending many messages to ensure the validity of data, therefore bandwidth is wide. Avoidance-based algorithms don't allow a reference opportunity about the stale data to maintain consistency. Because this consistency checks are delayed to the last, the decrease of the number of message transmission and rapid response are possible, but conflict is found late, which is increasing aborting rate. [Fra97, Fra96]. However, it is difficult for these techniques to apply to objects that have complex characteristics such as abstraction, encapsulation, polymorphism, inheritance [Wol96]. Therefore, the researches to maintain object consistency are need.

In this paper, we proposed two techniques for the maintenance of an object caching consistency based on RMI. The first technique is Time Stamp (TS) technique that compares update time in a server with cache time in a client when a client asked a server of consistency check to use a cached object. We applied TS to RMI. This modified RMI is called TS-RMI. The second is Invalid Message (IM) technique that broadcasts object updating message to all clients using this cached object when object is changed in server. This modified RMI which applies IM to RMI is called IM-RMI. We measured average response time of method invocation according to object update frequency under equal computing environments to compare TS-RMI with IM-RMI.

The rest of the paper is organized as follows. In section 2, we describe related work. The proposed consistency maintenance techniques are explained in section 3, and experimental results are presented in section 4. Finally, we conclude our work and suggest future work in section 5.

2. RELATED WORK

In fixed computing environment, data consistency maintenance techniques have been classified by detection-based and avoidance-based. At mobile computing environments, detection-based algorithms such as NWL-NH are used between a mobile host and a base station, avoidance-based algorithms such as O2PL are used between fixed hosts [Jin95], techniques such as AT, SIG periodically broadcast a client the fact that database is updated for consistency maintenance in disconnection [Bar94].

Detection-based algorithms permit stale data in local, examine the validity of a server when read or write operations are performed. In C2PL, synchronization has been needed. A server doesn't send updating message to a client and takes the responsibility for all locking and deadlock detection. 2PL techniques are expanded from client/server environments to centralized control database environments, whereas implementation is simple, Each time they make a server perform validity check process at read or write operations, therefore, message transmission frequency is high and bandwidth is narrow [Car91, Wan91]. NWL is an asynchronous technique. Message transmission frequency is low in NWL. It is different from C2PL in write operations, and it makes server check validity. NWL-NH is adapted at mobile environments. After server broadcasts updated data items to a client periodically, a client removes invalid data items in cache [Jin95]. AOCC is a detection-based optimistic technique that allows a transaction to access cached data. It defers validity check until the transaction commits phase. In AOCC, if a transaction aborts locally, the server need not be notified as all of the transactions are performed locally until the commit. Because it is incurred primarily in the client, it makes abort cost low and performance high [Bod04].

Avoidance-based algorithms don't allow a reference opportunity about the stale data to maintain consistency. CBL is locally cached page copies are always guaranteed to be valid, so transactions can read them without contacting the server (i.e., only a local read lock is required). On a cache miss, the client sends a page request message to the server. The server returns a valid copy of the requested page when it determines that no other active clients believe they have write permission for the page. In callback locking, write intentions are declared synchronously, a client must have write permission on a page before it can grant a local write lock to a transaction. Because write permissions are obtained during transaction execution, transactions can commit after completing their operations without performing any additional consistency maintenance actions [Fra97]. ACBL is a synchronous avoidance-based algorithm as it uses lock-escalation messages in a synchronous manner, it sends a request for lock-escalation and waits for a reply before proceeding [Bod04, Gru97, and Zah97]. AACC technique is that all client/server manage lock with page and object unit, read-lock divide by private-read lock and shared-read lock. Private-read lock is cached to one client and shared-read lock is cached to several clients. AACC has high performance than ACBL, and low aborting rate than AOCC [Tam98]. O2PL acquires read and write lock locally until transaction completion, examines accuracy to a server when there is completion. It shows that transmission messages are reduced as compared with C2PL [Fra96, Car91].

3. PROPOSED TECHNIQUES

Object caching consistency means that an original object in a server is equal to a cached object in a client. The proposed techniques applied object caching consistency maintenance problem to RMI. These modified RMI that apply proposal techniques in general RMI is called TS-RMI and IM-RMI. We need some hypotheses as follows. First, objects of old version may be in cache basically. Second, Remote method invocation is happening from many clients to optimum level frequently. Also, we put Cache Manager and Consistency Manager commonly in modified RMI and take charge processing about caching and consistency maintenance respectively.

3.1 Time Stamp Technique (TS-RMI)

This technique adds to general RMI time stamp function to compare client's cache time to server's update time for consistency. Client's cached object is changed into a new object that reflects server's update time recently. Time comparison is processed by client-initiated which try to use object. Consistency is maintained, when cache time is the greater than update time (⑧**Valid** of [Figure 1]). Therefore, Client's Cache Manager invokes a local object method. This process shows ①**-**⑩ in [Figure 1].

In other case, the object was stored in caching table before a remote object is changed. That is, it is state that consistency doesn't maintain (Ⓖ ***Invalid*** of [Figure 1]). At this time, Server's Consistency Manager invokes remote method, then it makes Skeleton transmit result and an object to Stub through network.

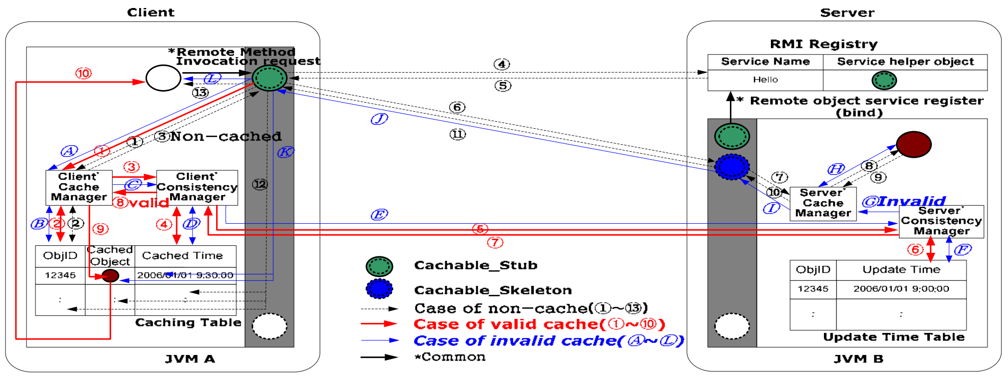

Figure 1. System Model of TS-RMI

Stub changes contents of caching table with a transmitted object (Update on Caching Table Ⓚ), returns result to a client object. This process shows Ⓐ-Ⓛ marked in [Figure 1].

When the remote method is invoked for the first time, there is no cached object in the caching table (③ Non-cached of [Figure 1]). In this case, consistency check is not necessary. Stub stores result and an object that transmitted from Skeleton in caching table (Add to Caching Table ⑫), returns result to client object. This process shows ①−⑬ marked in [Figure 1].

An advantage of TS-RMI is the fact that the consistency checked simply, the communication bandwidth is decreased by handling client-initiated consistency check. Also, server's responsibility for consistency maintenance is decreased because it is not necessary for server to send object update message.

3.2 Invalid Message Technique (IM-RMI)

In IM-RMI, as soon as a caching object is changed, a server broadcasts invalid message to clients which have ever used the remote object caching (*1-3* of [Figure 2]). Therefore, it must have information about all clients that possesses a cached object on broadcasting message table in a server. Whenever caching happens, Server's Cache Manager keeping client's IP-Address in own broadcasting message table.

On the other hand, a client maintains consistency by checking transmitted state of invalid message. Consistency is maintained when client did not receive invalid message (⑤ **Valid** of [Figure 2]), therefore, Client's Cache Manager invokes a local object method. This process shows ①-⑦ in [Figure 2].

In the case of receiving a invalid message, consistency is not maintained (Ⓔ ***Invalid*** of [Figure 2]), then Client's Consistency Manager makes Stub invoke a remote method, gets result and a object by Skeleton from Server's Cache Manager, stores updated object in client's caching table, and sets invalid message field as default value (zero) for next invocation (Update on Caching Table Ⓜ). This process shows Ⓐ-Ⓝ marked in [Figure 2].

When the remote method is invoked for the first time, there is no cached object in the caching table (③ Non-cached of [Figure 2]). In this case, consistency check is not necessary. Stub stores result and an object that is transmitted from Skeleton in caching table (Add to Caching Table ⑬), returns result to a client object. This process shows ①−⑭ marked in [Figure 2].

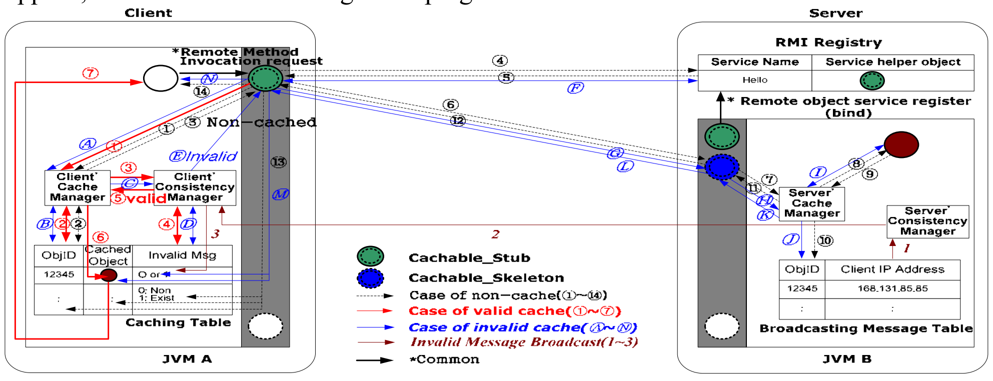

Figure 2. System Model of IM-RMI

ISBN 80-86943-12-7

An advantage of IM-RMI is the fact that the consistency check around time is decreased because client access to server only when client has received invalid message. Therefore, this RMI can get fast response time.

4. PERFORMANCE COMPARISON

We experimented with server such as Pentium IV 3.0GHz, Main Memory 2GB, Windows XP, Desktop Computer, Marvell Yukon 88E8001 PCI Gigabit Ethernet Controller, LAN 100Mbps, and client such as Pentium IV 1.6GHz, Main Memory 256MB, Windows XP, IBM Laptop Computer, Orinoco Wireless LAN PC Card (5volt), WaveLAN 11Mbps. Both a server and a client are installed in JDK 1.5.0_06.

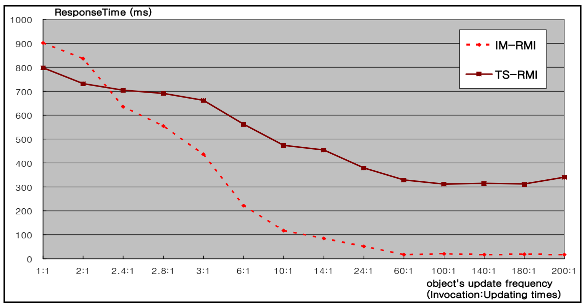

Figure 3. Average Response Time Comparison TS-RMI with IM-RMI

We measured response time respectively, while a method is invoked 30 times, object updated intervals are from 500 ms to 100,000 ms, and then compared TS-RMI with IM-RMI under equal computing environments. Elapsed time of remote method takes about 500 ms. In case an object update cycle takes 500 ms, whenever a method invoked, an object was changed (1:1). In case an object update cycle takes 100,000 ms, when a method is invoked 200 times, an object was changed only one (200:1).

As experimental results, the more object update is, TS-RMI is faster than IM-RMI in response time. Otherwise, IM-RMI is faster than TS-RMI in response time as shown in [Figure 3].

5. CONCLUSIONS

In this paper, we proposed two techniques for object consistency that has complex characteristics such as abstraction, encapsulation, polymorphism, inheritance. We experimented based on RMI. One is TS-RMI that compares update time of a server with cache time of a client when client checks consistency to server for using cached object. The other is IM-RMI that broadcasts an object updating message to all clients using this cached object when object is changed in server. As results, TS-RMI was efficient when the frequency of server object update is high, and IM-RMI was efficient that server's object update is infrequent. Therefore, modified RMI could apply selectively at the distributed applications considering object update frequency.

We have presented mechanisms for efficient caching consistency of objects for RMI applications. Using these mechanisms, caching can be easily and transparently added to existing RMI applications, while preserving RMI compatibility. The central mechanism includes the ability to support the Cache Manager and Consistency Manager. Stub and Skeleton class that generated by RMIC are modified by CRMIC. No changes are required to existing Client and Server applications. Also existing RMI performance can be enhanced without losing backward-compatibility. The future work of this study will be additional technique for mobile computing environments.

6. REFERENCES

[Bar94] Barbara D., Imielinski T., "Sleepers and Workaholics: Caching Strategies in Mobile Environments", ACM SIGMOD, pp. 1-12, 1994.

[Bod04] Bodorik P., Jutla D., Lu Y., "Interoperable Server-based Cache Consistency Algorithm", IEEE Database Engineering and Applications Symposium, 2004.

[Car91] Carey M., Franklin M., Livny M., Shekita E., "Data Caching Tradeoffs in Client-Server DBMS Architectures", ACM SIGMOD, pp. 357-366, 1991.

[Dow98] Downing T. B., "Java RMI: Remote Method Invocation", IDG Books, pp. 13-15, 1998.

[Fra96] Franklin M., Carey M., "Client Data Caching: A Foundation for High Performance Object Database System", Kluwer Academic Publishers, 1996.

[Fra97] Franklin M., Carey M., Livny M., "Transactional Client-Server Cache Consistency: Alternatives and Performance", ACM TODS, pp. 315-363, 1997.

[Gru97] Gruber R., "Optimism VS. Locking: A Study of Concurrency Control for Client Server Object-Oriented Databases", Ph.D Thesis, MIT, 1997.

[Jin95] Jin Jing et al., "Distributed Lock Management for Mobile Transaction", IEEE Distributed Computing System, 1995.

[Tam98] Tamer M., Kaladhar M., "An Asynchronous-Based Cache Consistency Algorithm for Client Caching DBMSs", VLDB, pp. 440-451, 1998.

[Wan91] Wang Y., Rowe L., "Cache Consistency and Concurrency Control in a Client/Server DBMS Architecture", ACM SIGMOD, pp. 367-376, 1991.

[Wol96] Wollrath A., Riggs R., Waldo J., "A Distributed Object Model for the Java System", 2nd Conference on Object-Oriented Technologies, 1996, Toronto, Ontario, Canada.

[Zah97] ZahariousDakis M., Franklin M., "Adaptive, Fine Grained Sharing in a Client-Server OODBMS: A Callback-Based Approach", ACM TODS, pp. 570-627, 1997.